Universidad de Buenos Aires

Facultad de Ingeniería

Módulo electrónico para laboratorio remoto

Documento de arquitectura de software

Raúl Emilio Romero
rraulemilioromero@gmail.com

23/11/2023
Versión A

Registros de cambios

Revisión	Detalles de los cambios realizados	Fecha
0	Creación del documento	23/11/23

Contenidos

1 Introducción

1.1 Propósito

1. Este documento representa una arquitectura de software para un módulo electrónico adquisidor de datos para laboratorios remotos (LR).

2. Está dirigido a desarrolladores que se ocupen del análisis, diseño e implementación, así como también a quienes desarrollen el testing de software.

1.2 Ámbito del sistema

1. Este software forma parte de un desarrollo más amplio llevado a cabo en el contexto de la carrera Maestría en Internet de las Cosas (MIoT) y cuyo objetivo es desarrollar un laboratorio remoto para la enseñanza de la física.

2. Este software llevará el nombre comercial de SACDMELR (Software de adquisición y control de datos para módulos electrónicos de laboratorios remotos).

3. Se comercializará integrado a laboratorios remotos destinados a la enseñanza de la física en carreras de ingeniería.

1.3 Definiciones, Acrónimos y Abreviaturas

BROKER	Servidor MQTT con el que se comunican los clientes
CEIoT	Carrera de Especialización en Internet de las Cosas
CESE	Carrera de Especialización en Sistemas Embebidos
HAL	Capa Abstracción de hardware
JSON	JavaScript Object Notation
LR	Laboratorio remoto
MIoT	Maestría en Internet de las Cosas
MQTT	Message Queuing Telemetry Transport
SACDMELR	Software de adquisición y control de datos para módulos electrónicos de laboratorios remotos
UART	Universal Asynchronous Receiver-Transmitter

1.4 Referencias

1. Plan de Proyecto del Trabajo Final de la Carrera Especialización en Internet de las Cosas. Ing. Raúl Emilio Romero. Plan de Proyecto del Trabajo Final.

2. Documento de especificación de requerimientos de software: MELR-ER-0001 versión A.

3. Documento de especificación de casos de uso: MELR-CU-0001 versión A.

1.5 Visión general del documento

1. Este documento incluye al inicio una definición del tipo de arquitectura utilizada.

2. Posteriormente se incluyen los componentes de software, sus responsabilidades e interfaces.

2 Arquitecura

2.1 Patrones

1. Para este software se emplearán cuatro patrones arquitectónicos:

    - Arquitectura en capas
    - Capa de abstracción de hardware (HAL)
    - Segmentación de procesos
    - Control ambiental

2.1.1 Arquitectura por capas

1. Este patrón es utilizado cuando se desea separar la funcionalidad del software por niveles de abstracción.

2. En este proyecto se emplearán las siguientes capas:

    - Capa de Aplicación
    - Capa de Sistema operativo
    - Capa Abstracción de hardware (HAL)

2.1.2 Capa de abstracción de hardware (HAL)

1. Este patrón es utilizado cuando se desea abstraer el acceso al hardware.

2. Debido a que en producción se utilizará el microcontrolador ESP32-C3 y durante el desarrollo el kit ESP32-C3-DevKitC-02, se empleará el firmware de Espressif ESP-IDF v5.1.1 como capa de abstracción de hardware.

2.1.3 Segmentación de procesos

1. Este patrón se usa para transformar datos de una representación a otra antes de que puedan procesarse.

2. Se emplerá para la codificación/decodificación de la información recibida/enviada en formato JSON al Broker MQTT y para la adquisición de datos de los sensores analógicos.

2.1.4 Control Ambiental

1. Este patrón se emplea cuando un sistema incluye sensores que proporcionan información sobre el entorno y los actuadores que pueden cambiar el entorno.

2. Se empleará este patrón arquitectónico para la capa aplicación.

2.1.5 Arquitecura propuesta

1. En la figura 1 se muestra la arquitectura propuesta, los componentes intervinientes y el flujo de la información entre ellos.

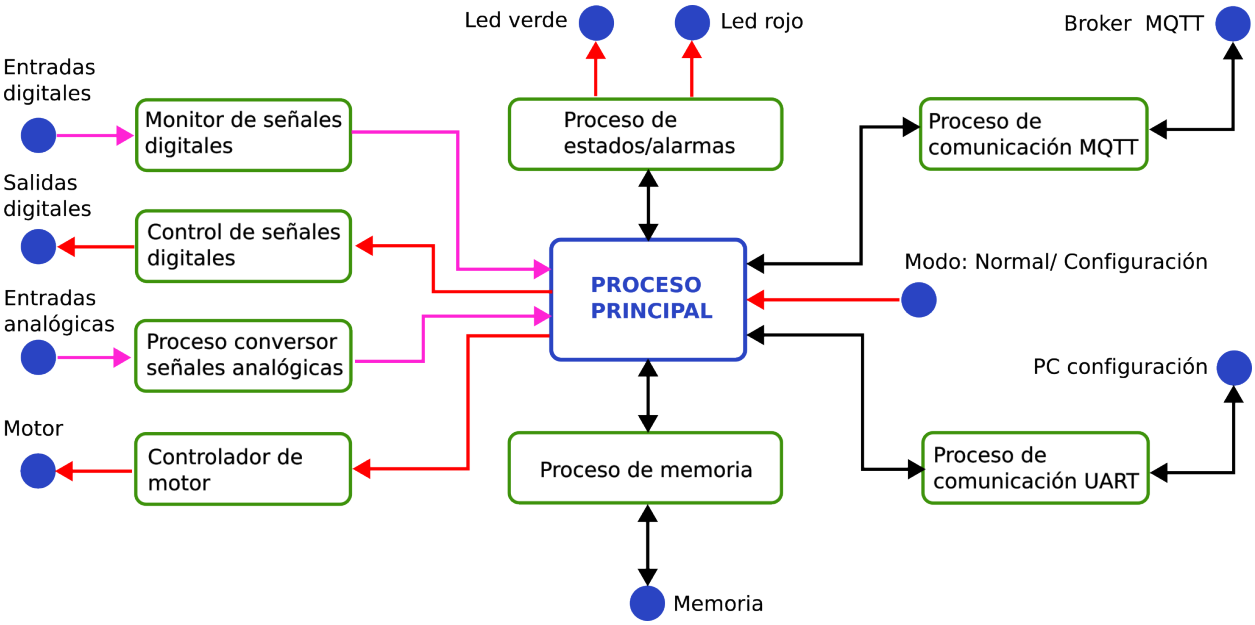

Figure 1: Arquitectura propuesta.

2.2 Componentes

1. Cada capa de software es considerada un componente de software. Con lo cual se poseen los siguientes componentes:

   - Aplicación
   - Sistema Operativo
   - HAL

2. A su vez, la capa de aplicación estará compuesta por los siguientes componentes de software:

   - Monitor de señales digitales: es el responsable de registrar los estados (alto o bajo) de las señales digitales conectadas al sistema.
   - Control de señales digitales: es el responsable de setear los estados (alto o bajo) de cada una de las salidas digitales que dispone el sistema.
   - Proceso adquisidor de señales analógicas: es el responsable de configurar, registrar y amacenar en un buffer las señales analógicas conectadas al sistema.
   - Controlador de motor: es el responsable de configurar y poner en marcha el motor conectado al sistema.
   - Proceso de estados y alarmas: es el responsable de manejar el estado de encendido de los leds rojo y verde según lo disponga el proceso principal. El led verde se enciende en forma intermitente si se reciben datos desde el Broker MQTT, mientras que el led rojo se enciende si el consumo eléctrico del módulo electrónico supera 1 A.
   - Proceso de memoria: es el responsables de almacenar/recuperar los datos de configuración realizados vía comunicación serie.
   - Proceso de comunicación MQTT: es el responsable de codificar/decodificar la información intercambiada con el Broker MQTT.

- Proceso de comunicación UART: es el responsable de codificar/decodificar la información intercambiada con la PC de configuración.
- Proceso principal: es el responsable de gestionar las tareas que deben realizar los componentes de software, a partir de los mensajes intercambiados con el Broker MQTT y/o vía PC de configuración.

2.3 Interfaces

1. Monitor de señales digitales:
   - Input: Señales digitales de niveles TTL. Ancho de banda 20 Hz. Cantidad de señales simultáneas 4.
   - Output: **bool [ ]** digitalInputGPIO
2. Control de señales digitales:
   - Input: **bool [ ]** digitalOutput
   - Output: Señales digitales de niveles TTL. Ancho de banda 20 Hz. Cantidad de señales simultáneas 4.
3. Proceso adquisidor de señales analógicas:
   - Input: Señales analógicas de rango ± 5V. Ancho de banda 1 kHz. Cantidad de señales simultáneas 4. Resolución 1 mV.
   - Output: **float [ ][ ]** analogSignal
4. Control de motores:
   - Input: **config_t[ ]** motorConfig
   - Output: Señales digitales TTL para la configuración del driver del motor (posible candidato A4988).
5. Proceso de estados y alarmas:
   - Input: **setState_t[ ]** setLEDstate
   - Output: Salidas digitales (pines GPIO del microcontrolador) para encender/apagar (de forma intermintente o continua) cada driver de cada led.
6. Proceso de memoria:
   - Input: **setConfigMEM_t** dataMem
   - Output: **getConfigMEM_t** dataMem
7. Proceso de comunicación MQTT:
   - Input: **receiveMQTTmsgdata_t** receiveMsgMqtt
   - Output: **sendMQTTmsgdata_t** sendMsgMqtt
8. Proceso de comunicación UART:
   - Input: **receiveUartCommand_t** receiveCommandUart

- Output: **sendUartCommand_t** sendCommandUart

9. Proceso principal: procesa, envía y recibe todos los datos generados/solicitados por el resto de los módulos de la arquitectura. Por medio de un switch conectado a un puerto GPIO del microcontrolador (input) el proceso principal se dispone en modo Normal o modo Configuración.